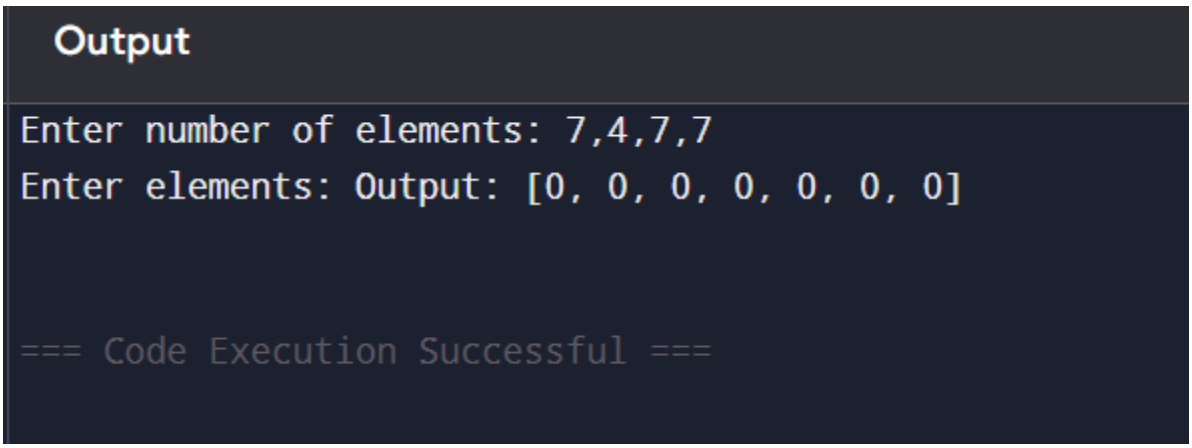
LAB 2

PROGRAM 1

```
#include <stdio.h>


void sort(int arr[], int n) {
    int i, j, temp;
    for(i = 0; i < n - 1; i++) {
        for(j = 0; j < n - i - 1; j++) {
            if(arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main() {
    int arr[100], n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter elements: ");
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    sort(arr, n);
```

```
    printf("Output: [");
    for(i = 0; i < n; i++) {
        printf("%d", arr[i]);
        if(i < n - 1) printf(", ");
    }
    printf("]\n");
    return 0;
}
```

OUTPUT

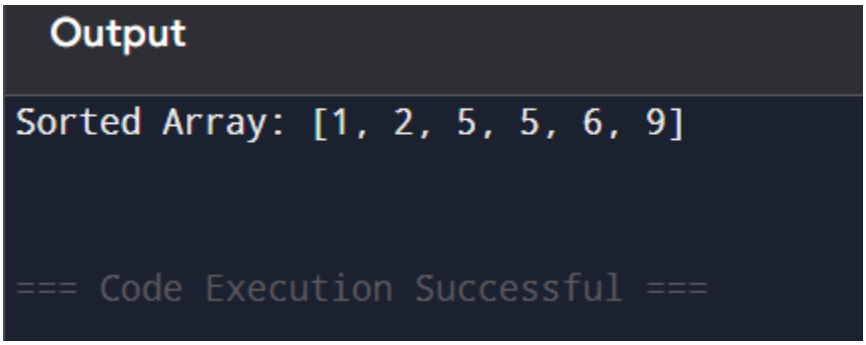


PROGRAM 2

```
#include <stdio.h>

void selectionSort(int arr[], int n) {
    int i, j, min, temp;
    for(i = 0; i < n - 1; i++) {
        min = i;
        for(j = i + 1; j < n; j++) {
            if(arr[j] < arr[min])
```

```
            min = j;
        }
        temp = arr[i];
        arr[i] = arr[min];
        arr[min] = temp;
    }
}

int main() {
    int arr[] = {5, 2, 9, 1, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    selectionSort(arr, n);

    printf("Sorted Array: [");
    for(int i = 0; i < n; i++) {
        printf("%d", arr[i]);
        if(i < n - 1) printf(", ");
    }
    printf("]\n");
    return 0;
}
```

OUTPUT

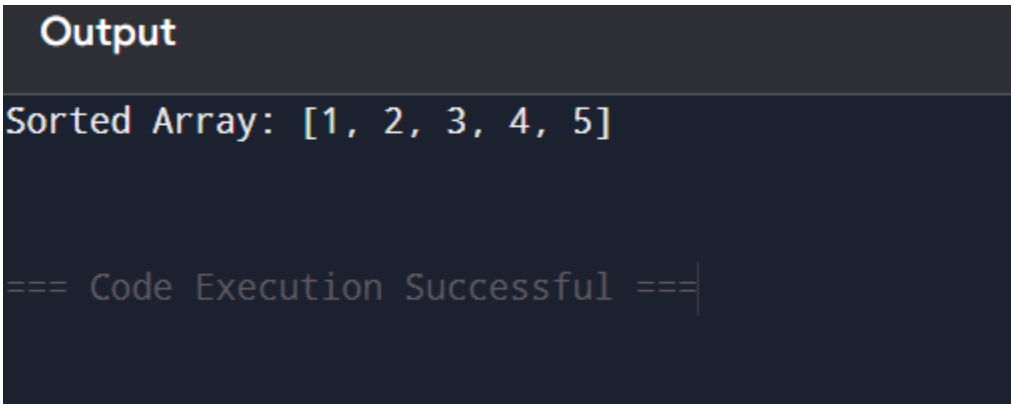


PROGRAM 3

```
#include <stdio.h>


void bubble_sort(int arr[], int n) {
    int i, j, temp;
    int swapped;

    for(i = 0; i < n - 1; i++) {
        swapped = 0;

        for(j = 0; j < n - i - 1; j++) {
            if(arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = 1;
            }
        }
```

```
        if(swapped == 0)
            break;
    }
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    bubble_sort(arr, n);
    printf("Sorted Array: [");
    for(int i = 0; i < n; i++) {
        printf("%d", arr[i]);
        if(i < n - 1) printf(", ");
    }
    printf("]\n");

    return 0;
}
```

OUTPUT

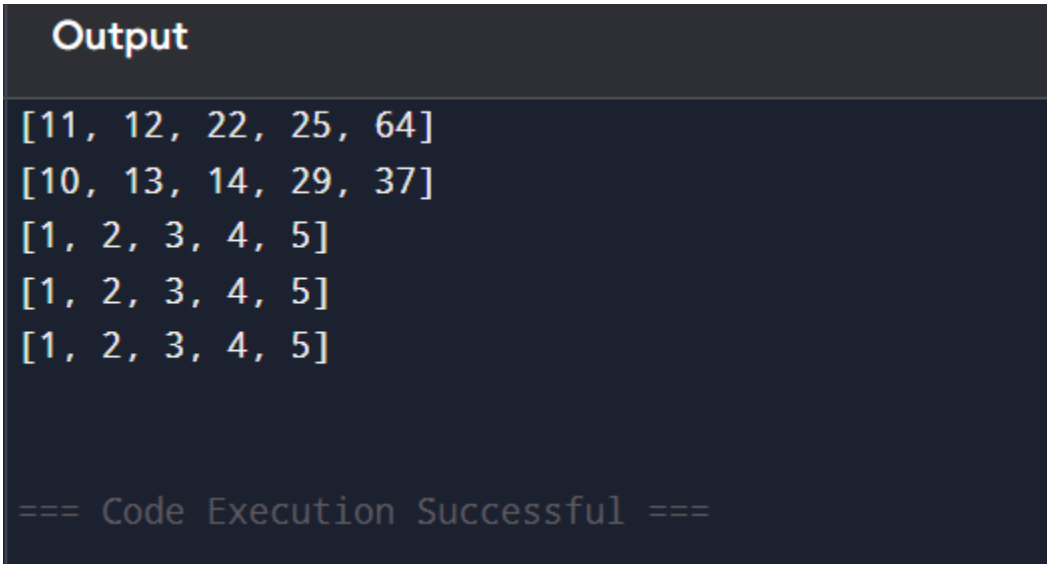


PROGRAM 4

```
#include <stdio.h>


void bubble_sort(int a[], int n){
    int i,j,t,sw;
    for(i=0;i<n-1;i++){
        sw=0;
        for(j=0;j<n-i-1;j++)
            if(a[j]>a[j+1]){
                t=a[j]; a[j]=a[j+1]; a[j+1]=t;
                sw=1;
            }
        if(!sw) break;
    }
}
void print(int a[],int n){
    printf("[");
    for(int i=0;i<n;i++){ printf("%d",a[i]); if(i<n-1) printf(", "); }
    printf("]\n");
}
int main(){
    int
t1[]={64,25,12,22,11},t2[]={29,10,14,37,13},t3[]={3,5,2,1,4},t4[]={1,2,3,4,5},t5[]={5,4,3,2,1};
    bubble_sort(t1,5); print(t1,5);
    bubble_sort(t2,5); print(t2,5);
    bubble_sort(t3,5); print(t3,5);
    bubble_sort(t4,5); print(t4,5);
```

```
    bubble_sort(t5,5); print(t5,5);
}
```

OUTPUT

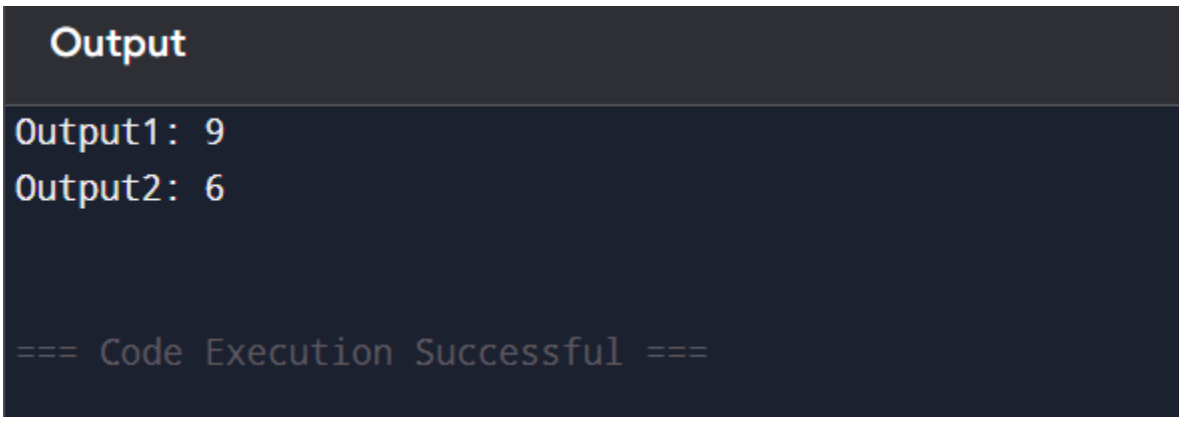


PROGRAM 5

```
#include <stdio.h>


int findKthMissing(int arr[], int n, int k) {
    int expected = 1, i = 0;
    while(k > 0) {
        if(i < n && arr[i] == expected)
            i++;
        else
            k--;
        expected++;
    }
    return expected - 1;
}
```

```
int main() {
    int arr1[] = {2,3,4,7,11}, k1 = 5;
    int arr2[] = {1,2,3,4}, k2 = 2;

    printf("Output1: %d\n", findKthMissing(arr1, 5, k1)); // 9
    printf("Output2: %d\n", findKthMissing(arr2, 4, k2)); // 6
    return 0;
}
```

OUTPUT

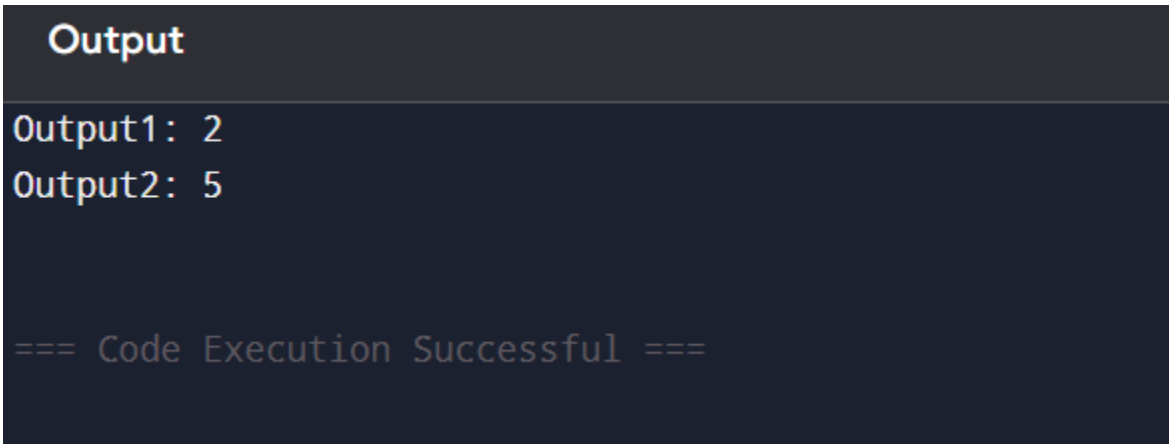


PROGRAM 6

```
#include <stdio.h>

int findPeakElement(int nums[], int n) {
    int left = 0, right = n - 1;
    while (left < right) {
        int mid = (left + right) / 2;
        if (nums[mid] > nums[mid + 1])
            right = mid;      // peak is on left (including mid)
```

```
        else
            left = mid + 1;   // peak is on right
    }
    return left;  // or right, both point to peak index
}

int main() {
    int nums1[] = {1, 2, 3, 1};
    int nums2[] = {1, 2, 1, 3, 5, 6, 4};

    printf("Output1: %d\n", findPeakElement(nums1, 4));  // 2
    printf("Output2: %d\n", findPeakElement(nums2, 7));  // 5 or 1
    return 0;
}
```

OUTPUT

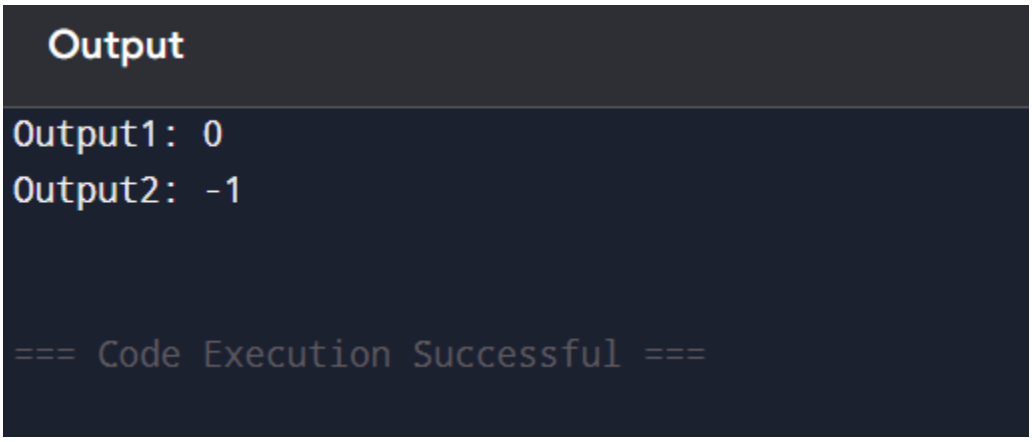


PROGRAM 7

```
#include <stdio.h>
#include <string.h>
```

```
int strStr(char haystack[], char needle[]) {
    int n = strlen(haystack);
    int m = strlen(needle);
    if(m == 0) return 0;

    for(int i = 0; i <= n - m; i++) {
        int j;
        for(j = 0; j < m; j++) {
            if(haystack[i + j] != needle[j])
                break;
        }
        if(j == m)
            return i;
    }
    return -1;
}

int main() {
    char hay1[] = "sadbutsad", needle1[] = "sad";
    char hay2[] = "leetcode", needle2[] = "leeto";

    printf("Output1: %d\n", strStr(hay1, needle1)); // 0
    printf("Output2: %d\n", strStr(hay2, needle2)); // -1
    return 0;
}
```

OUTPUT

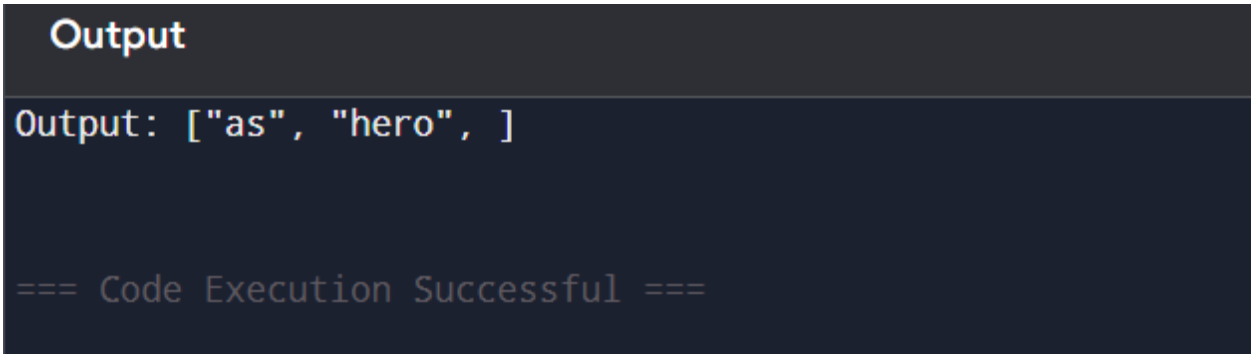


PROGRAM 8

```
#include <stdio.h>
#include <string.h>

int main() {
    char words[4][20] = {"mass", "as", "hero", "superhero"};
    int n = 4, found;
    printf("Output: [");

    for(int i = 0; i < n; i++) {
        found = 0;
        for(int j = 0; j < n; j++) {
            if(i != j && strstr(words[j], words[i])) {
                found = 1;
                break;
            }
        }
        if(found) {
            printf("\"%s\"", words[i]);
```

```
            if(i < n - 1) printf(", ");
        }
    }
    printf("]\n");
    return 0;
}
```

OUTPUT

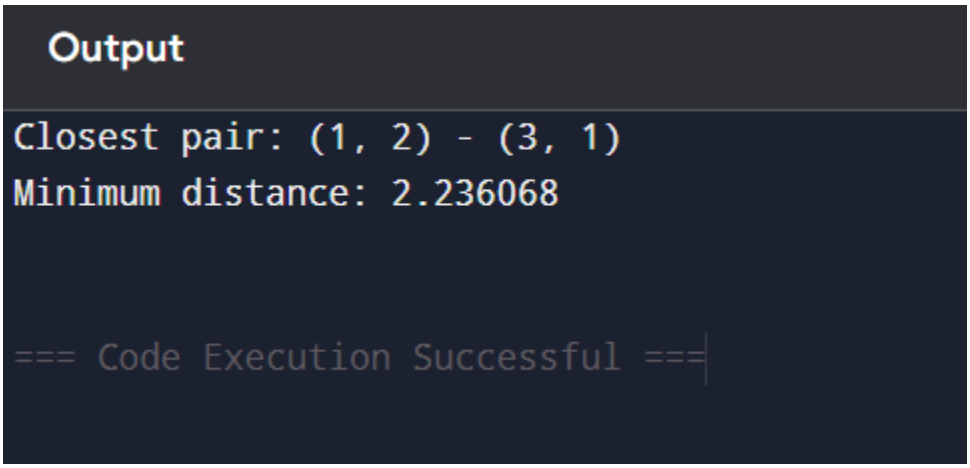


PROGRAM 9

```
#include <stdio.h>
#include <math.h>

typedef struct {
    double x, y;
} Point;

double distance(Point a, Point b) {
    return sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y));
}

int main() {
    Point pts[] = {{1,2}, {4,5}, {7,8}, {3,1}};
```

```
    int n = 4;
    double minDist = 1e9, d;
    Point p1, p2;

    for(int i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
            d = distance(pts[i], pts[j]);
            if(d < minDist){
                minDist = d;
                p1 = pts[i];
                p2 = pts[j];
            }
        }
    }
    printf("Closest pair: (%.0f, %.0f) - (%.0f, %.0f)\n", p1.x, p1.y, p2.x, p2.y);
    printf("Minimum distance: %lf\n", minDist);
    return 0;
}
```

OUTPUT



PROGRAM 10

```
#include <stdio.h>
#include <math.h>

typedef struct {
   double x, y;
} Point;
double distance(Point a, Point b) {
   return sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y));
}
void closestPair(Point pts[], int n) {
   double minDist = 1e9, d;
   Point p1, p2;
   for(int i=0;i<n;i++){
      for(int j=i+1;j<n;j++){
         d = distance(pts[i], pts[j]);
         if(d < minDist){
            minDist = d;
            p1 = pts[i];
            p2 = pts[j];
         }
      }
   }
   printf("Closest pair: (%.1f, %.1f) - (%.1f, %.1f)\n", p1.x, p1.y, p2.x, p2.y);
   printf("Minimum distance: %.4lf\n", minDist);
}
int orientation(Point a, Point b, Point c) {
```

```
    double val = (b.y - a.y)*(c.x - b.x) - (b.x - a.x)*(c.y - b.y);
    if (fabs(val) < 1e-9) return 0;
    return (val > 0) ? 1 : 2;
}

void convexHull(Point pts[], int n) {
    printf("Convex Hull Points: ");
    for(int i=0;i<n;i++) {
        for(int j=i+1;j<n;j++) {
            int pos=0, neg=0;
            for(int k=0;k<n;k++) {
                if(k==i || k==j) continue;
                double val = (pts[j].x - pts[i].x)*(pts[k].y - pts[i].y)
                        - (pts[j].y - pts[i].y)*(pts[k].x - pts[i].x);
                if(val > 0) pos++;
                else if(val < 0) neg++;
            }
            if(pos==0 || neg==0)
                printf("P(%.1f, %.1f) ", pts[i].x, pts[i].y);
        }
    }
    printf("\n");
}

int main() {
    Point pts[] = {{10,0}, {11,5}, {5,3}, {9,3.5}, {15,3}, {12.5,7}, {6,6.5}, {7.5,4.5}};
    int n = 8;
```

```
    closestPair(pts, n);
    convexHull(pts, n);

    return 0;
}
```

OUTPUT

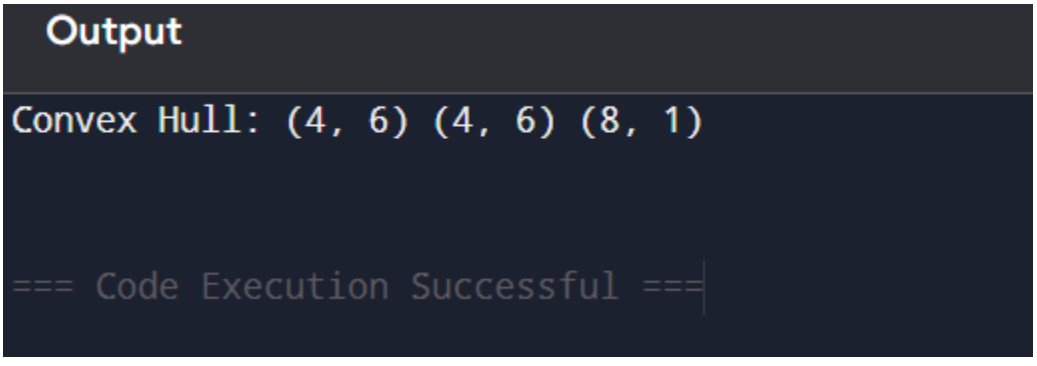
```
Closest pair: (9.0, 3.5) - (7.5, 4.5)
Minimum distance: 1.8028
Convex Hull Points: P(10.0, 0.0) P(10.0, 0.0) P(5.0, 3.0) P(15.0, 3.0) P(12
    .5, 7.0)


=== Code Execution Successful ===
```

PROGRAM 11

```
#include <stdio.h>
typedef struct {
    float x, y;
} Point;
int orientation(Point a, Point b, Point c) {
    float val = (b.y - a.y)*(c.x - b.x) - (b.x - a.x)*(c.y - b.y);
    if (val == 0) return 0;
    return (val > 0) ? 1 : -1;
}
void convexHull(Point pts[], int n) {
    printf("Convex Hull: ");
    for (int i=0;i<n;i++) {
        for (int j=i+1;j<n;j++) {
```

```
            int pos=0, neg=0;
            for (int k=0;k<n;k++) {
                if (k==i || k==j) continue;
                float val = (pts[j].x - pts[i].x)*(pts[k].y - pts[i].y)
                        - (pts[j].y - pts[i].y)*(pts[k].x - pts[i].x);
                if (val > 0) pos++;
                else if (val < 0) neg++;
            }
            if (pos==0 || neg==0)
                printf("(%.0f, %.0f) ", pts[i].x, pts[i].y);
        }
    }
    printf("\n");
}
int main() {
    Point pts[] = {{1,1}, {4,6}, {8,1}, {0,0}, {3,3}};
    int n = 5;
    convexHull(pts, n);
    return 0;
```

OUTPUT



PROGRAM 12

```
#include <stdio.h>
#include <math.h>
#include <float.h>

#define N 5   // Maximum number of cities

double distance(double a[2], double b[2]) {
   return sqrt((a[0]-b[0])*(a[0]-b[0]) + (a[1]-b[1])*(a[1]-b[1]));
}

void swap(int *a, int *b) {
   int temp = *a; *a = *b; *b = temp;
}

// Recursive function to generate permutations and find min distance
void tsp(double cities[][2], int path[], int start, int end, double *minDist, int bestPath[]) {
   if (start == end) {
      double dist = 0;
      for (int i=0; i<end; i++)
         dist += distance(cities[path[i]], cities[path[i+1]]);
      dist += distance(cities[path[end]], cities[path[0]]);  // Return to start

      if (dist < *minDist) {
         *minDist = dist;
         for (int i=0; i<=end; i++) bestPath[i] = path[i];
         bestPath[end+1] = path[0]; // return to start
```

```
        }
        return;
    }

    for (int i=start; i<=end; i++) {
        swap(&path[start], &path[i]);
        tsp(cities, path, start+1, end, minDist, bestPath);
        swap(&path[start], &path[i]);
    }
}

int main() {
    // ---- Test Case 1 ----
    double cities1[4][2] = {{1,2}, {4,5}, {7,1}, {3,6}};
    int n1 = 4;
    int path1[N], bestPath1[N+1];
    double minDist1 = DBL_MAX;

    for (int i=0; i<n1; i++) path1[i] = i;
    tsp(cities1, path1, 1, n1-1, &minDist1, bestPath1);

    printf("Test Case 1:\nShortest Distance: %lf\nShortest Path: ", minDist1);
    for (int i=0; i<=n1; i++)
        printf("(%.0lf, %.0lf) ", cities1[bestPath1[i]][0], cities1[bestPath1[i]][1]);
    printf("\n\n");

    // ---- Test Case 2 ----
```

```
    double cities2[5][2] = {{2,4}, {8,1}, {1,7}, {6,3}, {5,9}};
    int n2 = 5;
    int path2[N], bestPath2[N+1];
    double minDist2 = DBL_MAX;

    for (int i=0; i<n2; i++) path2[i] = i;
    tsp(cities2, path2, 1, n2-1, &minDist2, bestPath2);

    printf("Test Case 2:\nShortest Distance: %lf\nShortest Path: ", minDist2);
    for (int i=0; i<=n2; i++)
        printf("(%.0lf, %.0lf) ", cities2[bestPath2[i]][0], cities2[bestPath2[i]][1]);
    printf("\n");

    return 0;
}
```

OUTPUT

Output

```
Test Case 1:
Shortest Distance: 16.969112
Shortest Path: (1, 2) (7, 1) (4, 5) (3, 6) (1, 2)

Test Case 2:
Shortest Distance: 23.129950
Shortest Path: (2, 4) (6, 3) (8, 1) (5, 9) (1, 7) (2, 4)


=== Code Execution Successful ===
```

PROGRAM 13

```
#include <stdio.h>
#include <limits.h>
#define N 3

int cost(int a[], int c[N][N]) {
    int s=0; for(int i=0;i<N;i++) s+=c[i][a[i]]; return s;
}

void swap(int *x,int *y){int t=*x;*x=*y;*y=t;}

void perm(int c[N][N],int a[],int l,int r,int *min,int best[]){
    if(l==r){int s=cost(a,c);if(s<*min){*min=s;for(int i=0;i<N;i++)best[i]=a[i];}return;}
    for(int i=l;i<=r;i++){swap(&a[l],&a[i]);perm(c,a,l+1,r,min,best);swap(&a[l],&a[i]);}
}

int main(){
    int cost1[N][N]={{3,10,7},{8,5,12},{4,6,9}};
    int cost2[N][N]={{15,9,4},{8,7,18},{6,12,11}};
    int a[N]={0,1,2},b[N],min;

    printf("Test Case 1:\n");
    min=INT_MAX;perm(cost1,a,0,N-1,&min,b);
    printf("Optimal: ");for(int i=0;i<N;i++)printf("(W%d,T%d) ",i+1,b[i]+1);
    printf("\nTotal Cost:%d\n\n",min);

    printf("Test Case 2:\n");
```

```
    min=INT_MAX;perm(cost2,a,0,N-1,&min,b);
    printf("Optimal: ");for(int i=0;i<N;i++)printf("(W%d,T%d) ",i+1,b[i]+1);
    printf("\nTotal Cost:%d\n",min);
}
```

OUTPUT

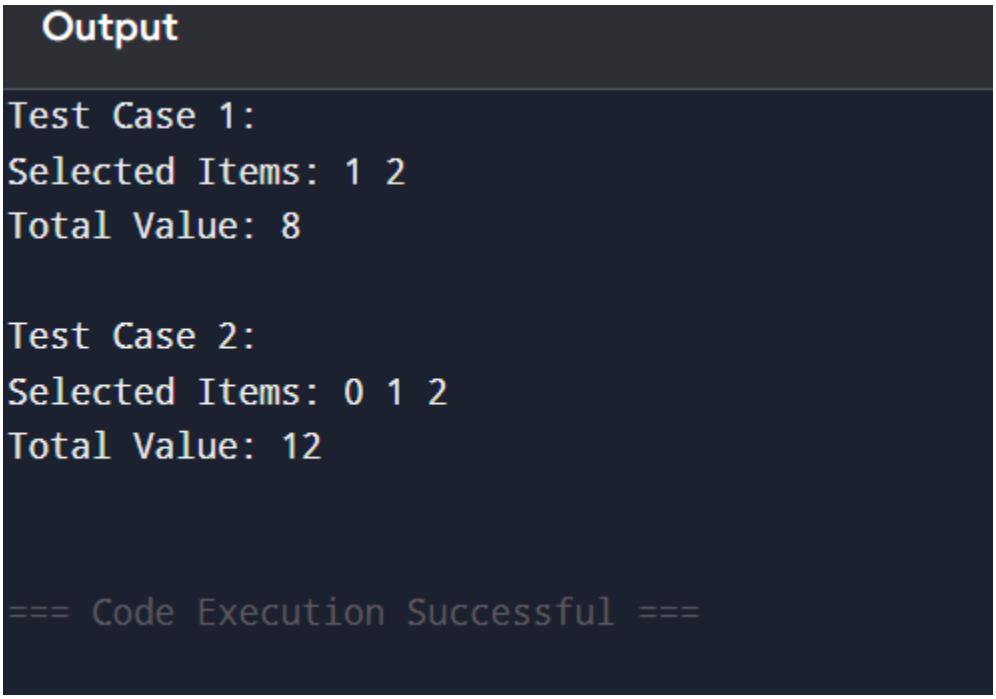
```
Output

Test Case 1:
Optimal: (W1,T3) (W2,T2) (W3,T1)
Total Cost:16

Test Case 2:
Optimal: (W1,T3) (W2,T2) (W3,T1)
Total Cost:17


=== Code Execution Successful ===
```

PROGRAM 14

```
#include <stdio.h>


int total_value(int items[], int values[], int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) if (items[i]) sum += values[i];
    return sum;
}


int total_weight(int items[], int weights[], int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) if (items[i]) sum += weights[i];
```

```
    return sum;
}

void knapsack(int n, int weights[], int values[], int cap) {
    int maxVal = 0, bestMask = 0;
    for (int mask = 0; mask < (1 << n); mask++) {
        int tw = 0, tv = 0;
        for (int i = 0; i < n; i++)
            if (mask & (1 << i)) { tw += weights[i]; tv += values[i]; }
        if (tw <= cap && tv > maxVal) { maxVal = tv; bestMask = mask; }
    }

    printf("Selected Items: ");
    for (int i = 0; i < n; i++) if (bestMask & (1 << i)) printf("%d ", i);
    printf("\nTotal Value: %d\n", maxVal);
}

int main() {
    int w1[] = {2,3,1}, v1[] = {4,5,3}, n1 = 3, c1 = 4;
    int w2[] = {1,2,3,4}, v2[] = {2,4,6,3}, n2 = 4, c2 = 6;

    printf("Test Case 1:\n");
    knapsack(n1, w1, v1, c1);

    printf("\nTest Case 2:\n");
    knapsack(n2, w2, v2, c2);
}
```

OUTPUT


Output
Test Case 1:
Selected Items: 1 2
Total Value: 8
Test Case 2:
Selected Items: 0 1 2
Total Value: 12
=== Code Execution Successful ===